Protocol Synthesis from Service Specifications Described by Graph Rewriting Rules(Theory of Rewriting Systems and Its Applications)

AUTHOR(S):

Takura, Akira; Sera, Takafumi; Ohta, Tadashi

数理解析研究所講究録
918巻 1995年 72-84

Protocol Synthesis from Service Specifications Described by Graph Rewriting Rules

Akira Takura Takafumi Sera Tadashi Ohta

ATR Communication Systems Research Laboratories
2-2 Hikaridai, Seika-cho, Soraku-gun, Kyoto 619-02 Japan

Abstract

A protocol synthesis method is proposed to derive communications software from service specifications of distributed systems. A communications service specification can be described as a set of graph rewriting rules. Each rule specifies a global state transition of multiple terminals associated with an event occurring at one of the terminals. The communications system is regarded as a black box in this description. A rule is represented by a pair of labeled directed graphs. The state of all the terminals in a communications system is also represented as a labeled directed graph. In this graph representation, a communications service specification describes a set of rules to find the appropriate graph representing the condition part of a rule; its resultant is substituted for an isomorphic subgraph in a graph representing a communications system. The isomorphic subgraph detection is known to be NP-complete. The derived distributed programs operate within a time practical for application because of the limited number of terminals usually associated with a communications service. Accordingly, the proposed software specification generation method can be used to develop communications software. The proposed software generation method can be applied to many kinds of distributed system based on state transition systems.

1 Introduction

Communications software generation from service specifications is a promising way to reduce the cost of developing reliable communications software. This paper proposes a communications software generation method from service specifications that can be described without detailed knowledge of communications systems or communications service implementation methods.

Many specification languages including SDL [1] and LOTOS [2] have been proposed for specifying communications software or protocols [3, 4]. When these languages are used to develop communications software, stepwise refinement of specifications is needed [5, 6, 7]. Cameron et al. [5] uses rule-based language L.0 for implementing a real-life protocol. Tsai et al. [6] uses frame-and-rule oriented requirement specification language FRORL. In these methods, specifications are incrementally refined to obtain protocol specifications; however, they could not synthesize protocol specifications from service specifications.

Bochmann and Gotzhein [8, 9], Chu and Liu [10], Saleh and Probert [11], and Kakuda et al. [12] proposes protocol synthesis methods from service specifications. In these methods, all actions at service access points and their execution orders are described. In this paper, a specification description language STR (State Transition Rule) [13] is used for describing communications service specifications. In STR communications service specifications can be described by specifying terminal behavior which can be recognized from outside of a communications system. With this method, we can describe specifications without detailed knowledge of the target communications system. A service is defined as a set of production rules. Consequently, we do not have to describe the execution order of events at service access points.

STR assumes that a communications system consists of homogeneous processes that correspond to ter-

minals. Processes sharing a relationship form a global state. Each STR rule specifies a global state transition of terminals for an input event that has occurred at a terminal. In this service specification description, terminals are defined as service access points in a communications system. Since communications systems are geographically distributed systems, we generate a communications protocol that can operate using local state of terminals when global state transitions are given.

The state of a terminal connected to a communications system is represented by a subgraph of a labeled directed graph that represents the state of all the terminals. Using this graph representation, an STR rule can be regarded as a graph rewriting rule consisting of an initial graph and a next graph. Kawata et al. [14] proposes a protocol synthesis method with a restriction: in the initial graph of an STR rule, there is a path going through all labeled vertices. This restriction is imposed for the sake of reducing the number of communications between processes. The synthesized protocol makes to-and-fro communication among vertices in an initial graph. This paper presents a new method that obtains local state transition systems from service specifications described by global state transition rules. The synthesized protocol makes to-and-fro communication among vertices in an initial graph; however, there is no restriction imposed on initial graphs.

The proposed protocol synthesis method from service specifications takes a step toward automated software generation. Communications software can be obtained from synthesized protocol specifications by using a stepwise refinement method [15].

Section 2 defines communications service specifications and protocol synthesis. Section 3 briefly reviews STR. In section 4 we define a problem to be solved. Section 5 proposes a protocol synthesis method. Finally we show an example of the proposed protocol synthesis method.

2 Service Specifications and Protocol Synthesis

2.1 Communications Software Model

Protocol synthesis generates error-free protocols from service specifications. Most protocol synthesis methods assume a layered architecture as in Fig. 1. A communications system provides communications services for users who access the system through service access points SAP1, $\cdots$, SAP n. Within the communications system, protocol entities cooperate to provide services by exchanging messages. This communication between protocol entities is provided by the service provider of lower layers. In communications services including telephone services terminals correspond to SAP in Fig. 1.

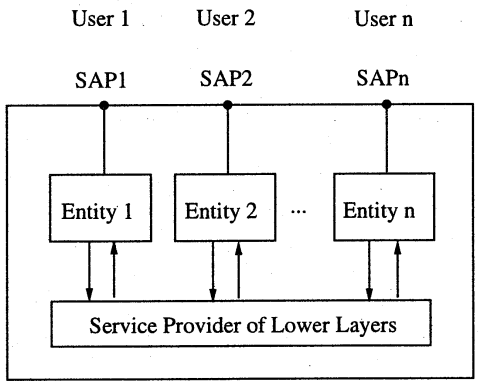

Figure 1: Layered architecture model

2.2 Service and Protocol Specifications

Bochmann, Gotzhein [8, 9], Chu et al. [10], and Saleh et al. [11] propose protocol synthesis methods assuming the relation between services and protocols illustrated in Fig. 1. Kakuda et al. [12] propose a protocol synthesis method that includes simultaneous occurrence of multiple events at different SAPs. A communications system provides users with communications services through SAPs. In this modeling, service specifications and protocol specifications are defined as follows [16]:

- The *service specification* describes what services the protocol entities of the lower protocol layer provide for users in the upper protocol layer. The services provided by the lower protocol layer are based on a set of service primitives that describe the operations at service access points through which the services are provided.

- The *protocol specification* describes interactions among protocol entities of the lower protocol layer. Interactions are defined in terms of services provided to the upper protocol layer and services available from the communication medium.

Hereafter, a protocol entity is called a process.

3 Service Specification Language

3.1 STR

A service is defined as a set of STR rules. An STR rule has the form:

initial state event: next state.

The "initial state" and the "next state" represent global states of terminals. A global state is represented by a set of local states. A local state is represented by a set of state primitives. A state primitive may have two arguments to express terminals. The first argument represents the terminal having the state primitive. If the second argument is specified, the terminal designated by the first argument holds a relation of the primitive to the terminal specified by the second argument. Therefore, the local state of a terminal is defined as the set of state primitives whose first argument designates the terminal. A state primitive represents a terminal state which is recognizable from outside of a communications system.

The "event" may also have two arguments to express terminals. It represents a logical input to the terminal designated by the first argument. If the second argument is described in an event, it represents a terminal identifier given by the event.

3.2 Graph Representation

We use graph representation of an STR rule to generate communications protocol from service specifications described with STR. An STR rule can be represented by two graphs. The initial graph corresponds to the initial state and an event; the next graph corresponding to the next state. Both an initial and a next graphs are called rule graphs. Figure 3 shows the graph representation of the rule described in Fig. 2.

A rule graph consists of a set of vertices and directed edges. Each vertex has its own name and some vertices have labels. A vertex is denoted by a circle. The name of a vertex is written in a circle, and the label of a vertex is written near the circle. Each edge has labels that are written near it. An initial graph has a label that shows an event. A vertex designated by the first argument of this label is called an event vertex.

A vertex that has a label or an edge incident from it is called *labeled.* Other vertices are called *unlabeled.* For each vertex in an initial graph, there must be a path from the vertex where an event occurred.

dial-tone(A), idle(B)
dial(A,B):
ring-back(A,B), ringing(B,A).

Figure 2: An example of an STR rule

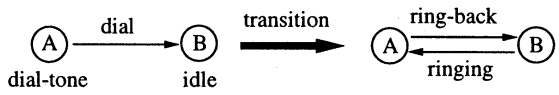

Figure 3: Graph representation of an STR rule

A global state for all the processes in a communications system is called a *system state*; the graph denoting the system state is called a *system graph.*

3.3 Application of STR Rules

According to an STR rule, in a system graph a subgraph that is isomorphic to the initial graph of the rule should be replaced by the next graph of the rule. Figure 4 illustrates how an STR rule is applied. If there are two rules $r1$ and $r2$ such that the initial graph of $r1$ is isomorphic to a subgraph of the initial graph of $r2$, then $r2$ is superior to $r1$. We denote $r1 < r2$ iff $r2$ is superior to $r1$. This is not total-order. Thus, there still exists the possibility that multiple rules may be applied. When multiple rules can be applied, a rule may be arbitrarily selected.

4 Problem

Let $g = (V, E, v^0)$ be a rooted labeled directed graph g that has a set of vertices V, a set of directed edges E and a root vertex v^0. For each vertex of a rooted directed graph, there is a path from v^0 to the vertex. The set of vertices of g is denoted by $V(g)$. The set of edges of g is denoted by $E(g)$. The root of g is denoted by $root(g)$. A labeled directed graph g is denoted by $g = (V, E)$ with a set of vertices V and a set of directed edges E.

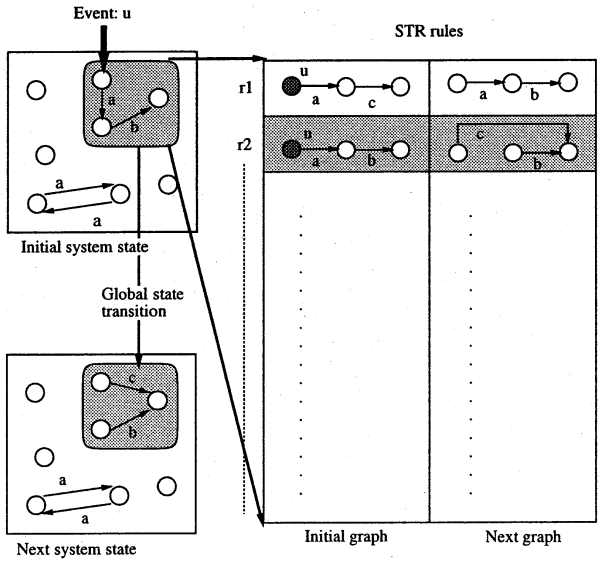

Figure 4: Application of STR rules

Definition 1 (Subgraph isomorphism)
Let $g = (V, E)$ be a labeled directed graph and $g_1 = (V_1, E_1, v_1^0)$ be a rooted labeled directed graph. The graph g contains a subgraph $g' = (V', E', v^0)$ isomorphic to g_1 if and only if there exist subsets $V' \subset V$ and $E' \subset E$ such that $v^0 \in V'$, $|V'| = |V_1|$, $|E'| = |E_1|$, and there is a one to one mapping $f : V_1 \longrightarrow V'$ that satisfies the following conditions.

$$f(v_1^0) = v^0$$
$$(x, y) \in E_1 \Longrightarrow (f(x), f(y)) \in E'$$
$$(f(x), f(y)) \in E' \Longrightarrow (x, y) \in E_1$$
$$\alpha(x) \subset \alpha(f(x))$$
$$\beta((x, y)) \subset \beta((f(x), f(y)))$$

where α is a function to get a set of labels attached on a vertex and β is a function to get a set of labels attached on an edge.

If g has a subgraph isomorphic to g_1, then we write $g_1 \sqsubset g$.

Definition 2 (Spanning path)
For a rooted labeled directed graph g, a spanning path $sp(g)$ is defined as a path that satisfies the following conditions.

1. *The vertices of $sp(g)$ is the set of labeled vertices in g.*
2. *The root of $sp(g)$ is $root(g)$.*
3. *If (u, v) is an edge of $sp(g)$, then every vertex between $root(g)$ and u as well as between $root(g)$ and v appears before u in $sp(g)$.*

The problem to be solved is formally defined as follows.

Definition 3 (Problem)
Let $R = \{r_1, ..., r_n\}$ be a set of STR rules. Let $G = \{g_1, ..., g_n\}$ be a set of initial graphs of R. Let $G' = \{g_1', ..., g_n'\}$ be a set of next graphs of R. Let $g = (V, E)$ be a system graph.

Find a pair of graphs $(sg(v^0, g), g_i)$ such that $sg(v^0, g)$ is isomorphic to $g_i \in G$, and there is no graph isomorphic to $g_k \in G$ such that g_i is isomorphic to a subgraph of g_k. Then change $sg(v^0, g)$ to be isomorphic to $g_i' \in G'$.

The following lemma is satisfied for subgraph isomorphism. We use this lemma to construct a distributed algorithm.

Lemma
Let $g = (V, E)$ be a labeled directed graph and $g_1 = (V_1, E_1, v_1^0)$ be a rooted labeled directed graph. Let p be a spanning path of g_1 with the set of edges $\{(u_1, u_2), \cdots, (u_{m-1}, u_m)\}$ where $u_1 = v_1^0$, and $V_1 = \{u_1, \cdots, u_m, u_{m+1}, \cdots, u_n\}$. The graph g contains a subgraph $g' = (V', E', v^0)$ isomorphic to g_1 if and only if $|V'| = |V_1|$, $|E'| = |E_1|$, and there is a mapping $f : V_1 \longrightarrow V'$ that satisfies the following conditions.
For any labeled vertices u_i, u_j in V_1 and unlabeled verices u_k, u_l in V_1,
$f(u_1^0) = v^0$,
$u_i \neq u_j \Longrightarrow f(u_i) \neq f(u_j)$,
$f(u_i) \neq f(u_k)$,
$u_k \neq u_l \Longrightarrow f(u_k) \neq f(u_l)$,
$(u_i, u_j) \in E_1 \Longleftrightarrow (f(u_i), f(u_j)) \in E'$,
$\alpha(u_i) \subset \alpha(f(u_i))$
$\beta((u_i, u_j)) \subset \beta((f(u_i), f(u_j)))$
$\beta((u_i, u_k)) \subset \beta((f(u_i), f(u_k)))$

This lemma implies that we can determine applicability of a rule by traversing along its spanning path.

5 Definitions

Protocol synthesis is to generate a distributed algorithm to find an isomorphic subgraph in a system graph. Each protocol entity holds a local state of a terminal. Then protocol synthesis is defined to derive protocol entity specifications with local states

from communications service specifications described by global state transition rules.

When a protocol entity receives an input from a terminal or some other protocol entity, it determines a rule to be applied or sends another protocol entity a request to inquire a surrounding global state determining an applicable rule. This communication for state inquiry is perfomed sequentially. Finally a rule is determined. The rule is informed the protocol entities that are inquired their states. Then each protocol entity changes its state according to the rule.

A spanning path is determined for each rule. The communication for state inquiry goes along a spanning path. If there are isomorphic subgraphs in multiple initial graphs, it is determined at the same time whether a system graph contains a subgraph isomorphic to the subgraphs. This reduces communication between protocol entities. For this purpose, we utilize a spanning path of a tree generated by overlapping spanning trees of the initial graphs of an event. This spanning path is called a provisional communication path. A synthesized protocol communicates along provisional communication paths. We give these definitions. Note that sometimes a vertex and a process are used for the same meaning.

Let C(e) denote a set of initial graphs with an event e.

Rule inclusion graph Let s be a connected subgraph that has an event vertex and is isomorphic to subgraphs of more than one element in C(e). Let D(s) denote the set of elements of C(e) that have s as their subgraphs. An ordered pair $< s$, D(s) $>$ is a vertex of the rule inclusion graph of C(e) if a graph s' generated from s by adding an edge is a subgraph of an element of C(e), and D(s) $\neq$ D(s'). An ordered pair $< r$, D(r) $>$ is also a vertex of the rule inclusion graph of C(e) if $r \in$C(e). There is an edge from $< s$, D(s) $>$ to $< t$, D(t) $>$ iff (1) $s \sqsubset t$, and (2) there is no other element $< u$, D(u) $>$ such that $s \sqsubset u$ and $u \sqsubset t$.

Rule inclusion graphs are used for determining orders of inquiring rule applicability. If there is an edge from $< s$, D(s) $>$ to $< t$, D(t) $>$ in a rule inclusion graph, then graph s is examined before graph t whether they are included in a system graph.

Figure 5 represents initial graph examples. Figure 6 shows the rule inclusion graph for the set of initial graphs in Fig. 5. The rule graph $r1$ is the maximum subgraph common to the rule graphs $r2$ and $r3$, and the maximum subgraph of the rule graph $r4$ is the rule graph $r2$. Consequently, there are four vertices $< r1$, $\{ r1, r2, r3, r4 \} >$, $< r2$, $\{ r2, r4 \} >$, $< r3$, $\{ r3 \} >$ and $< r4$, $\{ r4 \} >$ in this rule inclusion graph.

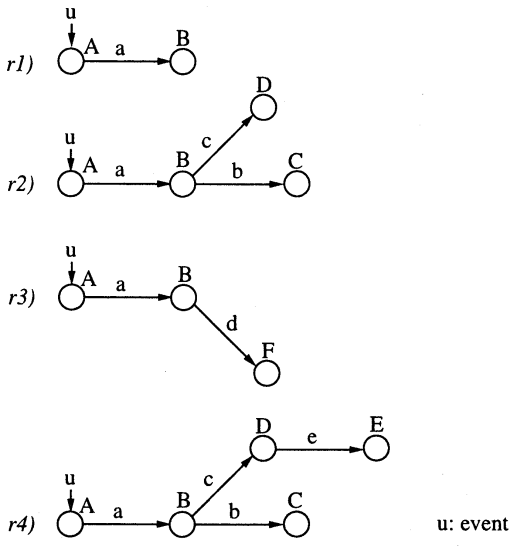

Figure 5: Example of initial graphs

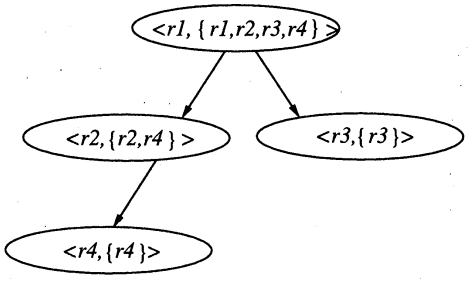

Figure 6: Example of a rule inclusion graph

Rule covering tree Let $g_r = (v_r, e_r, v^0)$ be the initial graph of a rule r. Let $t_r = (w_r, f_r, v^0)$ be a spanning tree of g_r. A rule covering tree c_r is defined as follows:

The set of vertices of c_r is v_r, and $root(c_r) = v^0$.

The edges from the same vertex are arranged clockwise by lexicographic order of edge labels. The set of edges of c_r is the union of e_r and the edges satisfying the following two conditions:

1. If a leaf vertex of t_r has a label, there exists an edge whose initial vertex is the leaf vertex of t_r.

2. An edge ab of c_r has a label of an ordered pair. The first element of the label is the label of a in v_r, and the second element is the label of ab in e_r.

A rule covering tree is uniquely determined for a spanning tree of an initial rule graph. A rule covering tree must satisfy the following two conditions for defining a rule overlapping tree. Let s be a spanning tree of the initial graph for rule r, and t a rule covering tree generated from s.

(1) s is an unlabeled subgraph of the initial graph for rule r.

(2) Let u be a vertex of s. Assume $g \sqsubset h$ for the two vertices g and h in the rule inclusion graph that has a vertex whose first element is r. If u is a vertex of h, and u is not a vertex of g, then an edge from one of the vertices in g to u is included.

Figure 7 shows rule covering trees for the initial graphs in Fig. 5.

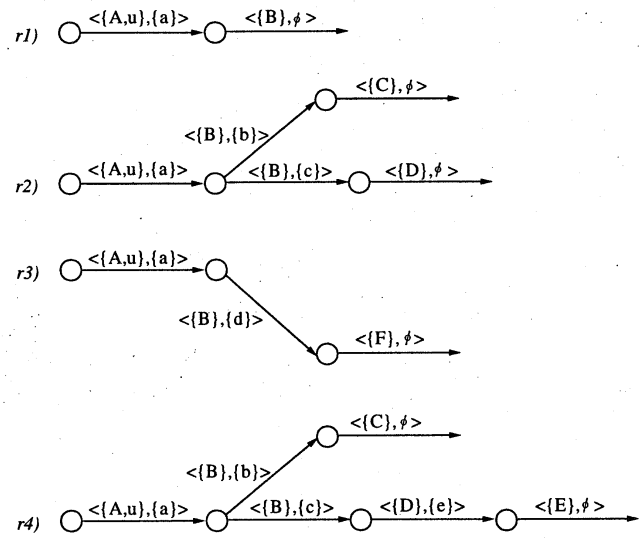

Figure 7: Example of rule covering trees

Rule overlapping tree A rule overlapping tree for C(e) is generated by overlapping graphs in C(e). The tree satisfies the following conditions.

1. All the roots of the initial graphs in C(e) is overlapped.

2. For each element g in C(e), there is a subgraph isomorphic to g.

3. If vertices u and v in rule covering trees s and t, respectively, are overlapped, then every vertex u' between $root(s)$ and u is overlapped to the vertex v' in t satisfying that its depth is the same as that of u' and v' is a vertex between $root(t)$ and v.

4. If deges e and f are overlapped, then the labels of e and f are the same or the one is included in the other.

5. The overlapping is performed in the lexicographic order of the labels of edges with the same initial vertex.

A rule overlapping tree is used for examining at the same time whether the common subgraphs in multiple rule graphs are included in a system graph.

Provisional communication path A provisional communication path for C(e) is defined as a path whose vertices is the vertices in the rule overlapping tree of C(e) such that:

1. The initial vertex is the event vertex.

2. Let g and h be two graphs constituting the first elements of two vertices in the rule inclusion graph for C(e). If $g \sqsubset h$, u is a vertex in both g and h; if v is not a vertex of g but a vertex of h, then u appears before v.

3. Let h be a graph corresponding to a vertex in the rule overlapping graph for C(e). For any graph g corresponding to a vertex in the rule inclusion graph for C(e) such that $g \sqsubset h$, assume that u and v are vertices in h, but not in g. In h, if u is a vertex in the path from the event vertex to v, or there is a vertex w in the path from the event vertex to u such that w is in the path from the event vertex to v and w is not a vertex in g, then u appears before v.

4. A unique label is attached to each edge.

We note that the vertex sequence of the provisional communication path for C(e) includes the sequence of a spanning path for each element of C(e) as a subsequence. It implies that state inquiry along a provisional communication path may determine a rule to be applied.

Inquiry message An inquiry message is a unique message for each edge between adjacent vertices

in a provisional communication path. In the synthesized protocol, an inquiry message has the following information in addition to its message name.

Actual communication path An actual communication path is a provisional communication path whose vertices are actual process identifiers to be inquired. Each process decides which process to send an inquiry message by using this information.

Temporary decided rule The rule with the highest priority in the rules satisfying their rule application conditions, i.e. their initial graphs are included in the system graph.

Rule candidates The remaining rules to be checked for applicability. When a process receives an inquiry message it screens rule candidates included in a received inquiry message by checking its local state.

Connection information Process information necessary for connection tests.

Visited processes A sequence of visited processes after an event occurred.

Temporary process sequence A process sequence for the current temporary decided rule.

Branches A branch consists of a process identifier and an inquiry message. When a process receives an inquiry message, the process compares its state with a subgraph of the rule overlapping tree corresponding to the inquiry message for obtaining a new inquiry message and branches (i.e. actual communication paths). This subgraph is the intersection of the rule overlapping tree and rule covering trees corresponding to rules contained in the rule candidates of the inquiry message. If plural actual communication paths to be inquired are obtained, the remaining actual communication paths except arbitrary one are stored as elements of inquiry messages in branches. Process identifiers are determined as the processes to which the obtained inquiry messages are sent.

A provisional communication path indicates inquiry messages used for examining whether rule graphs are included in a system graph, and their communication paths. An inquiry message usually implies multiple rules to be checked whether they are included in a system graph.

Figure 8 shows the rule overlapping tree for the initial graphs in Fig. 5 and its provisional communication path.

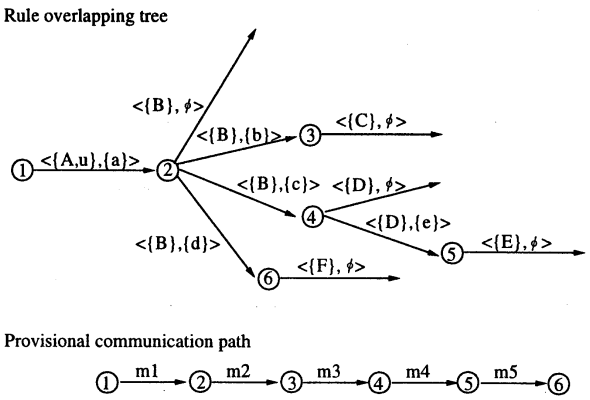

Figure 8: Example of a rule overlapping tree

Response message A message denoting whether to apply a rule or nothing at all. When a rule is indicated, it contains information which process the received message corresponds to.

Figures 9 and 10 show actual communication paths when the graphs are included in a system graph. The dotted arrows represent communication between processes and the inquiry messages attached to them.

In the example of Fig. 9, inquiry messages are communicated along a provisional communication path. Process A sends B a message m1($r1$, $r2$, $r3$, $r4$) that implies rules $r1$, $r2$, $r3$ and $r4$ are rule candidates. When B receives it, B determines rule $r1$ is included in the system graph, and rules $r2$, $r3$, and $r4$ are to be checked for their applicability. Then B sends C a message m2($r2$, $r3$, $r4$). When C receives the message m2($r2$, $r3$, $r4$), C sends m3($r2$, $r3$, $r4$) to D that is included in the received message as an actual communication path. In the end E receives a meesage m4($r3$, $r4$) and determines that rule $r4$ is included in the system graph. Since $r4$ is not included in any other initial graph, E detemines that rule $r4$ is to be applied. Then E sends processes A, B, C and D a response message indicating that each process changes its state according to rule $r4$. E also changes its state.

In the example of Fig. 10, inquiry message are communicated along a provisional communication path but some intermediate processes are skipped. When B receives a message m1($r1$, $r2$, $r3$, $r4$), B determines

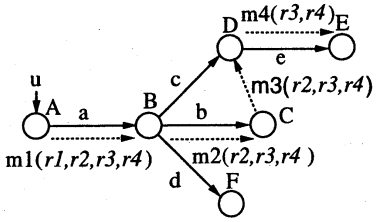

Figure 9: Example of communication along a provisional communication path

that rule $r1$ is a temoporary determined rule, rules $r2$ and $r4$ are not included in the system graph, and rule $r3$ is to be checked its applicability. In the end F determines rule $r3$ is the rule to be applied.

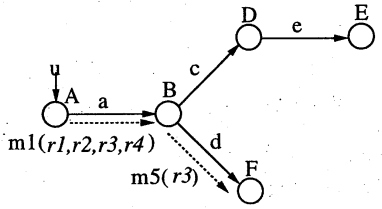

Figure 10: Example of communication skipping intermediate processes

Figure 11 shows an example of non-tree initial graphs. Figure 12 shows the rule overlapping tree and its provisional communication path for non-tree initial graphs in Fig. 11. Figure 13 illustrates provisional communication paths and inquiry messages for the rules in Fig. 11.

State transition segment One or two state transition segments are generated for each labeled vertex in the initial graph of a rule.

- When a vertex is the last vertex in a provisional communication path, a state transition segment is generated such that it receives a predetermined inquiry message in the provisional communication path; it sends a response message and changes to the next state determined by a rule.
- When a vertex is not the last vertex in a provisional communication path, two separate state transition segments are generated. One segment receives a predetermined inquiry message in the provisional communication path, sends another inquiry message along the provisional communication path, and then changes to the next state. The other segment receives a response message that includes a determined rule and changes to the next state.

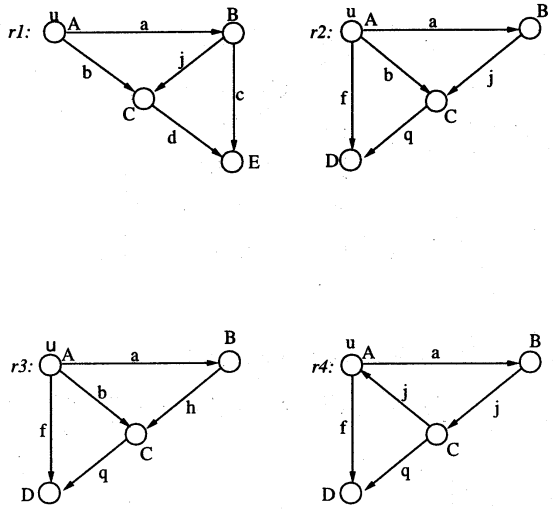

Figure 11: Example of non-tree initial graphs

Connection test A connection test is used to identify graphs that include plural vertices in a provisional communication path. Let i and j be two vertices in a provisional communication path, and i is nearer to the event vertex than j. A connection test decides if the edges from i excluding the edge in the provisional communication path are connected to vertices j or vertices connected by edges from j.

Figure 14 shows state transition segments of a vertex with the label B obtained from messages in the provisional communication path. Connection tests are omitted in this figure.

Figure 15 shows two initial graphs that need a connection test to distinguish them. The provisional communication path in these two initial graphs is the path 1,2,3. Figure 16 shows subgraphs that can be identified by processes 1 and 3. The two initial graphs in Fig. 15 are distinguished when process 3 receives a message that edge b is connected to process 4.

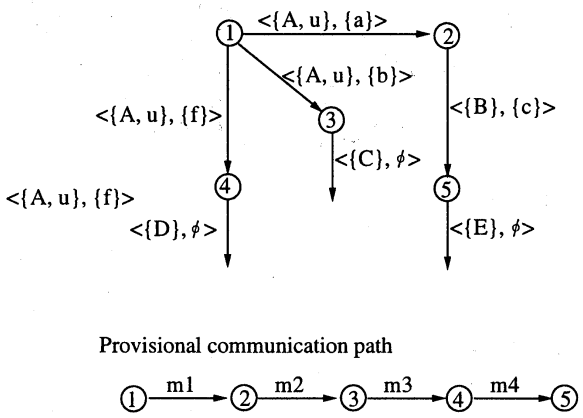

Figure 12: Example of a rule overlapping tree for complicated rules

6 Protocol Synthesis

The protocol synthesis algorithm consists of the following six steps.

(1) Classification of rule graphs

Rules are classified into sets of rules for the same event. Let C(e) be the set of a rule graph with an event e. For each rule graph set we initially apply steps 2 to 5, finally, step 6 is applied.

(2) Generation of rule inclusion graph

Generate a rule inclusion graph for each rule graph set.

(3) Generation of rule overlapping tree

Generate a rule overlapping tree.

(4) Determination of provisional communication path and inquiry messages

Determine a provisional communication path and inquiry messages from a rule overlapping tree.

(5) Generation of state transition segments

Generate state transition segments for each vertex of a rule covering tree corresponding to each rule r. The initial state of a state transition segment is the first element of the label attached to an edge incident from the vertex in the rule covering tree.

(6) Synthesis of process specification

Synthesize a state transition segment from the initial state until no new state is generated. Assume a new state s is generated. Collect all state transition segments whose initial state is included in s as a subgraph. The collected state transition segments are synthesized as follows.

- If two kinds of state transition segments correspond to the final vertex of an initial graph in the provisional communication path, a middle vertex is included, and their received messages are the same, the following synthesis occurs.

  The next states of these state transition segments are changed to the same states as their initial states. In the synthesized process specification, the rule candidates of the inquiry message to be sent are the intersection of the rule candidates of the received inquiry message and the set of rules whose state transition segments are synthesized. Other information is obtained as described in the above preliminaries.

  - When the resulting rule candidate set is not empty, the synthesized process sends the obtained inquiry message to the nearest process in the provisional communication path following the rules contained in the inquiry message.
  - When the resulting rule candidate set is empty and the branches in the received

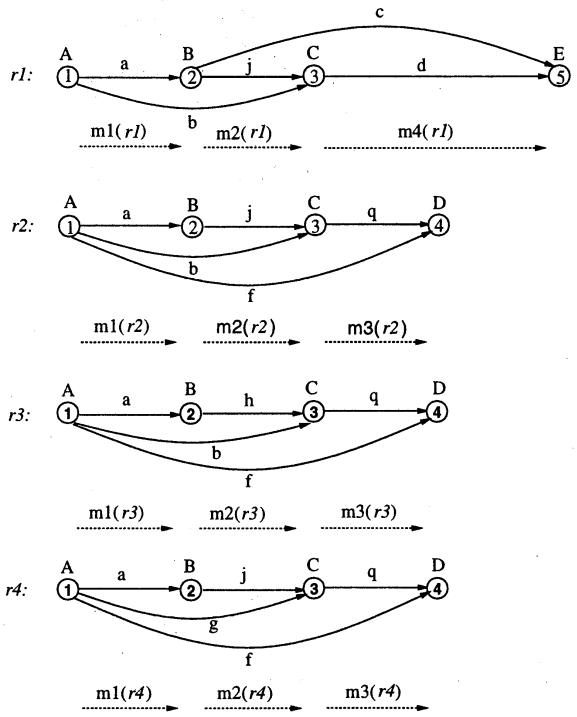

Figure 13: Communication for identifying rules in Fig. 11

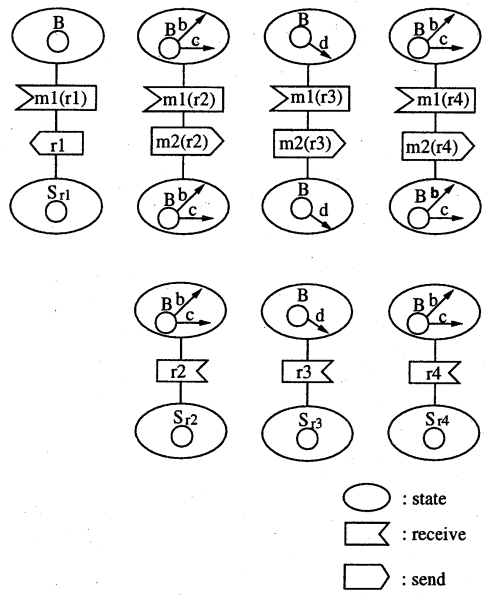

Figure 14: State transition segments

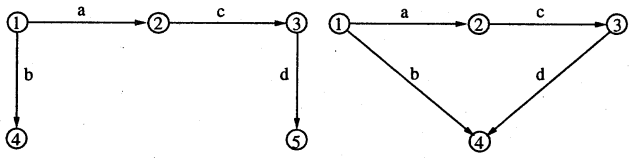

Figure 15: Two graphs requiring a connection test for identification

message are empty, two possible conditions exist: if the temporary decided rule is contained in the received inquiry message, the rule should be applied; if the temporary decided rule is not contained, a special response message is sent to all the visited processes.

- When the resulting rule candidate set is empty but the branches in the received message is not empty, the process sends an inquiry message to find a more superior rule than the temporary decided rule.

- If the above does not occur, the following happens.

  The collected state transition segments are synthesized as they are. Each state transition segment is synthesized as described above. Figure 17 illustrates how collected state transition segments are synthesized into a process specification.

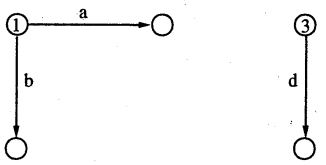

Figure 16: Subgraphs identifiable with local states

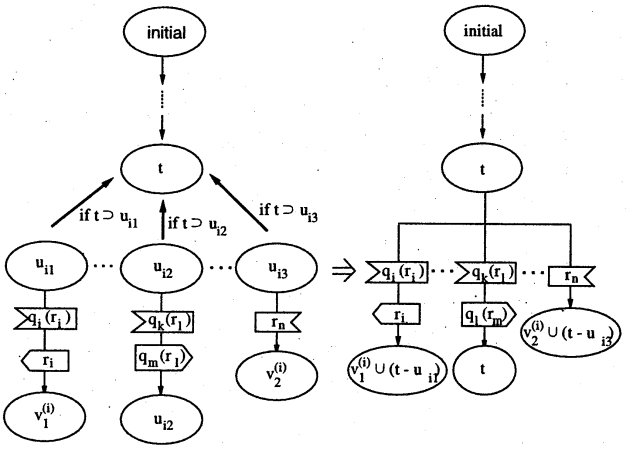

Figure 17: Synthesis of state transition segments

In the synthesized protocol specification, the rule candidates are screened as communication progresses, and then a rule to be applied is determined. Once a rule is determined to be applied at a process, the process sends response messages indicating the determined rule to the visited processes. There are two types of response messages: a message indicating a rule and no rule to be applied. A process that receives a response message changes to the state designated by the state transition segment corresponding to the determined rule. A process that receives a no rule message returns to the state prior to receiving the inquiry message.

The above protocol synthesis algorithm is implemented as a communications software generation sys-

tem. We have implemented some services on a PBX by using automatic generated software.

7 Example

We explain an example of process specification generation. Figure 18 expresses a service specification. In this service we need two terminals "data sender terminal" and "data receiver terminal". Data sender terminal starts data transmission by an event "start" when both the sender and the receiver which is specified by "start" are in the state "idle". The sender can always stop sending data by an event "stop". The receiver can always request the sender to pause sending data by an event *pause* and to resume to sending data by an event "resume". Figure 19 shows a global state transition diagram for this data sending protocol with pause function.

```
rule 1) idle(A),idle(B) start(A,B):
          sending(A,B),receiving(B,A).
rule 2) sending(A,B),receiving(B,A) stop(A):
          idle(A),idle(B).
rule 3) receiving(A,B),sending(B,A) pause(A):
          r-wait(A,B),s-wait(B,A).
rule 4) r-wait(A,B),s-wait(B,A) resume(A):
          receiving(A,B),sending(B,A).
```

Figure 18: STR description for data sending protocol

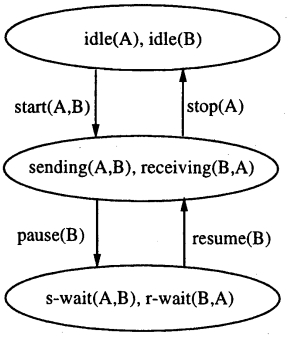

Figure 19: Global state transition diagram for data sending protocol with pause function

We show the graph representation for the service specification in Fig. 18.

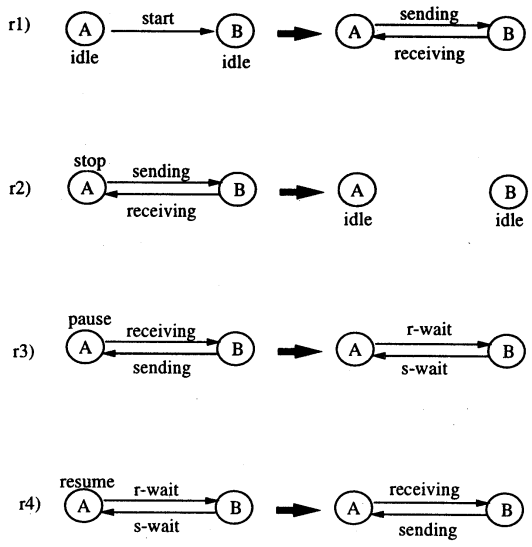

Figure 20: Graph representation of STR description for data sending protocol

In this example each rule makes a rule overlapping tree. Using this graph representation and the rule overlapping trees, we can get state transition segments which are parts of the objective protocol entity specification. Figure 21 shows state transition segments for the STR rules in Fig. 18. The messages used in inter-process communication are generated from the provisional communication paths.

Figure 22 shows a protocol entity specification synthesized from state transition segments in Fig. 21. In Fig. 22 the messages "m1", "m2", "m3", "m4" represent request message, the messages "r1", "r2", "r3", "r4" response message. In this specification actions to send or receive"norule" are omitted. "Norule" is a special response message to indicate there is no rule to be applied. In the generated protocol entity specification, when the protocol entity receives an unexpected request message, the protocol entity is assumed to send the message "norule" to the sender of the request message.

8 Conclusion

A protocol synthesis method is proposed by representing communications service specifications by a set of graph rewriting rules. The synthesized protocol implements a distributed algorithm that finds and re-

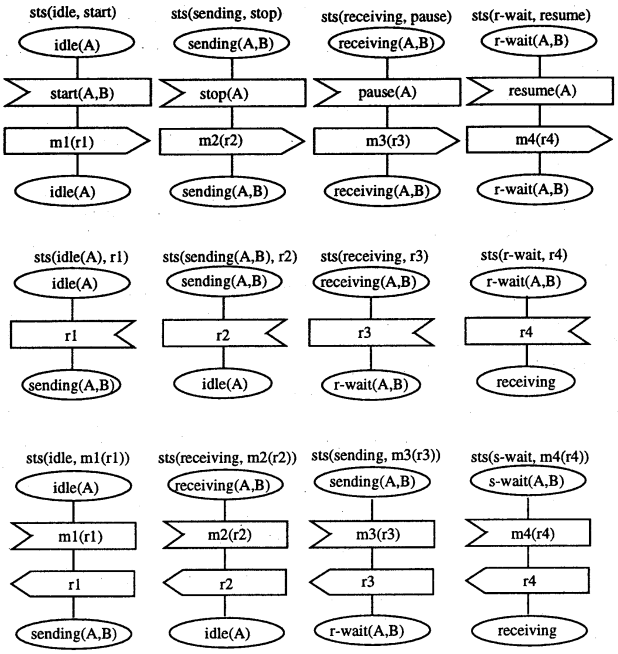

Figure 21: State transition segments for data communication protocol

places a subgraph isomorphic to an initial rule graph in a system graph with the next graph. Isomorphic subgraph detection is known as one of NP-complete problems [17]; however, the generated protocol operates within a time practical for application because of the limited number of terminals usually associated with a communications service.

The proposed protocol synthesis method can be used to generate communications software from service specifications. This enables those who are not-specialists in communications systems to participate in developing communications software.

Acknowledgments

We sincerely thank Dr. Kohei Habara, Chairman of the Board of ATR Communication Systems Research Laboratories, for his guidance and encouragement in this research. In addition to Dr. Nobuyoshi Terashima, President of ATR Telecommunication System Research Laboratories, we also wish to thank our colleagues for their helpful discussions.

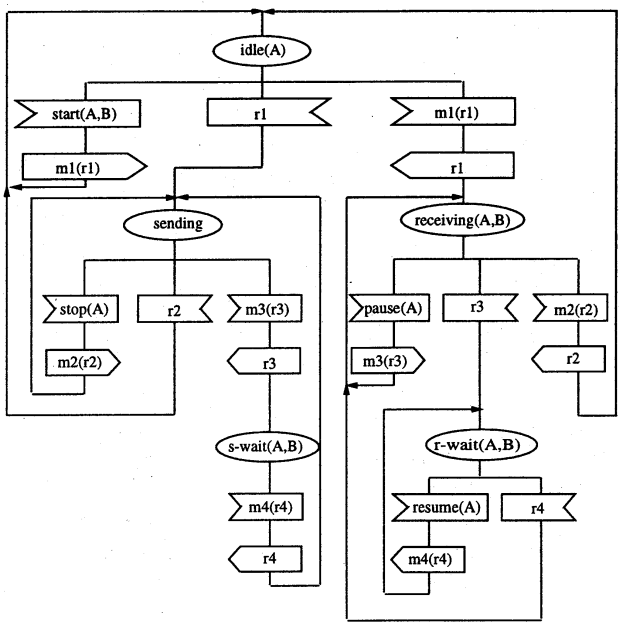

Figure 22: Protocol entity specification for data communication protocol

References

[1] CCITT, revised Recommendation Z.100, "CCITT Specification and Description Language (SDL)," May 1992.

[2] T. Bolognesi and E. Brinksma, "Introduction to the ISO Specification Language LOTOS," *Comput. Networks ISDN Syst.*, vol. 14, pp. 25–59, 1987.

[3] M. Faci, L. Logrippo, and B. Stepien, "Formal Specification of Telephone Systems in LOTOS: the Constraint-Oriented Style Approach," *Comput. Networks ISDN Syst.*, vol. 21, pp. 53–67, 1991.

[4] L. Drayton, A. Chetwynd, and G. Blair, "Introduction to LOTOS through a worked example," *Comput. Commun.*, vol. 15, no. 2, pp. 70–85, Mar. 1992.

[5] E. J. Cameron, D. M. Cohen, T. M. Guithner, W. M. Keese Jr., L. A. Ness, C. Norman, and H. N. Srinidhi, "The L.0 Language and Environment for Protocol Simulation and Prototyping," *IEEE*

Trans. on Comput., vol. 40, no. 4, pp. 562–571, Apr. 1991.

[6] J. J. P. Tsai, T. Weigert, and H.-C. Jang, "A Hybrid Knowledge Representation as a Basis of Requirement Specification and Specification Analysis," *IEEE Trans. on Software Eng.*, vol. 18, no. 12, pp. 1076 - 1100, Dec. 1992.

[7] C. Dendorfer and R. Weber, "From Service Specification to Protocol Entity Implementation - An Exercise in Formal Protocol Development," *Proc. IFIP Twelfth Int. Symp. Protocol Specification, Testing, Verification*, pp. 163–177, 1992.

[8] G. v. Bochmann and R. Gotzhein, "Deriving Protocol Specifications from Service Specifications," *Communications, Architectures & Protocols, Proc. ACM SIGCOMM '86* (Vermont, USA), pp. 136–145, Aug. 1986.

[9] R. Gotzhein and G. v. Bochmann, "Deriving Protocol Specifications from Service Specifications Including Parameters," *ACM Trans. Comput. Systems*, vol. 8, no. 4, pp. 255–283, Nov. 1990.

[10] P. M. Chu and M. T. Liu, "Synthesizing Protocol Specifications from Service Specification in the FSM Model," in *Proc. Comput. Networking Symp.*, pp. 505–512, Apr. 1988.

[11] K. Saleh and R. L. Probert, "A Service-Based Method for the Synthesis of Communications Protocols," *Int. J. Mini and Microcomput. Special Issue on Distributed Systems*, vol. 12, no. 3, pp. 197–103, 1990.

[12] Y. Kakuda, M. Nakamura, and T. Kikuno, "Automated Synthesis of Protocol Specifications from Service Specifications with Parallelly Executable Multiple Primitives," *IEICE Trans. Fundamentals*, vol. E77-A, no. 10, Oct. 1994.

[13] Y. Hirakawa and T. Takenaka, "Telecommunication Service Description Using State Transition Rules," *Proc. Sixth Int. Workshop on Software Specification and Design* (Como, Italy), pp. 140–147, Oct. 1991.

[14] K. Kawata, A. Takura, and T. Ohta, "On a Communication Software Generation Method from Communication Service Specifications Described by a Declarative Language," *Proc. Fifth International Conference on Computing and Information* (Sudbury, Canada), pp. 116–122, May 1993.

[15] A. Takura, K. Kawata, T. Ohta, and N. Terashima, "Communication Software Generation Based on Two-Layered Specifications and Execution Environment," *IEEE GLOBECOM'93* (Houston, Texas), pp. 362–368, Dec. 1993.

[16] M. T. Liu, "Protocol Engineering," *Advances in Computers*, vol. 29, Academic Press, 1989.

[17] M. R. Garey and D. S. Johnson, *Computers and Intractability, A Guide to the Theory of NP-Completeness.* NY: W. H. Freeman and Co., 1979.